JeDEM 8(3): 61-79, 2016
ISSN 2075-9517
http://www.jedem.org

Implementing Broadcast Channels with Memory for Electronic Voting Systems

Severin Hauser*, Rolf Haenni**

*University of Fribourg, Bern University of Applied Sience, Switzerland, severin.hauser@bfh.ch
**Bern University of Applied Sience, Switzerland, rolf.haenni@bfh.ch

Abstract: To provide universal verifiability, cryptographic voting protocols often require a broadcast channel to spread the election data to the public. The basic requirements for such a broadcast channel are similar for most protocols, for example that the channel maintains a memory of all broadcast messages and that nothing can be deleted from the channel's memory. In this paper, we provide a formal definition for such broadcast channels with memory and describe their properties. We also analyze the significance of a broadcast channel with memory in cryptographic voting protocols and propose that such a channel is provided in the form of a service that we call "bulletin board". Based on this service, we analyze some real-world problems that cryptographic voting protocols might have and provide possible solutions. For this we define a generic interface for the main board functionalities, which offers a flexible way of extending the basic properties of a bulletin board to comply with all sorts of additional requirements.

Acknowledgments: This research has been supported by the Hasler Foundation (Project No.14028).

Keywords: Electronic Voting, Verifiability, Public Bulletin Board, Broadcast Channel, Cryptography

1. Introduction

The concept of a bulletin board (BB) is an important building block to achieve universal verifiability in cryptographic voting protocols. Using a bulletin board in the context of electronic voting was first proposed by Cohen and Fischer (Cohen and Fischer 1985). They describe the bulletin board as a restricted shared memory, which can be read by everyone but only written to by the owner. Deletion or modification of messages is forbidden. From a conceptual point of view, a BB is often regarded as a broadcast channel with memory (BCM), which is a theoretical model of an ideal channel with very specific properties (Cramer, Gennaro, and Schoenmakers 1997a; Peters 2005). In the literature, the existence of a BCM is often assumed without providing a detailed definition of what a BCM is and without specifying its properties. The lack of proper definitions is

a problem for the general understanding of the cryptographic protocols and for analyzing their security properties.

In a real-world implementation of a given cryptographic protocol, the theoretical model of a BCM can at best be approximated. A common approach is to substitute the broadcast channel with memory by one or multiple additional parties participating in the protocol. These parties provide the service of accepting and memorizing the messages transmitted over the broadcast channel during the protocol execution. The service offered by such parties is what we call a bulletin board. It guarantees that messages cannot be deleted or modified, and it keeps track of the order in which the messages appeared. A bulletin board offering these two basic properties—called append-only bulletin board—is a prerequisite in almost every cryptographic voting protocol. In addition, some protocols require designated board sections for all involved parties (Cramer, Gennaro, and Schoenmakers 1997b), while other protocols require that the board rejects messages that are not well-formed (Haenni and Koenig 2013). When implementing a BB, appropriate solutions for such protocol-specific requirements need to be provided in addition to the append-only property (Hauser and Haenni 2016).

In a practical application of a BB in real elections, the situation gets even more complicated. There are numerous operational problems, for example, separating the election data when multiple elections run in parallel, opening and closing the electronic ballot box, archiving the election data when the protocol terminates, or ensuring consistent views for all users reading data from the board. This leads to highly specific implementations of bulletin boards, essentially one for each voting system. In practice, since building an electronic voting system is a very complex and time-consuming process on its own, the development of an appropriate BB with all desired properties has sometimes been given minor priority. There are several systems with a simplified BB based on a conventional data base system with a public interface. In such systems, the security of the cryptographic protocol gets undermined by an insecure BB implementation.

1.1. Contribution and Paper Overview

There are two main contributions of this paper. The first contribution is a consistent set of formal definitions for various types of channels, including a definition for a broadcast channel with memory (Section 2). To the best of our knowledge, our definitions are more concise and more comprehensive than any of the existing definitions in the literature. Our definitions describe how such channels behave under ideal circumstances. As such, they serve as a guideline for the design of bulletin boards, which mimic the ideal behavior of corresponding channels under real-world circumstances.

The second main contribution is an extended design proposal for a generic public interface of a flexible BB (Section 4.1). A first version of such a proposal has been published in (Hauser and Haenni 2016). The interface consists of two basic operations, one for posting new messages to the board and one for querying the board's current content. The generic specification of the interface allows these operations to be customized according to the actual requirements of a specific cryptographic protocol. While voting protocols are the primary target application for a BB

implemented according to our design, there are no limitations for applications of our approach in other fields.

An additional contribution of this paper is a compilation of possible BB properties, which we think might be useful in different contexts (Section 4.2 to 4.4). In particular, we think that a BB equipped with these properties can facilitate the implementation of voting protocols. We derive these properties from the abstract voting protocol introduced in Section 2.2. Our analysis exposes some of the practical problems that may arise with such protocols. To overcome these problems, we introduce additional BB properties and show how to implement them using the generic interface.

1.2. Related Work

The idea of publishing the election data on a public bulletin board has a long tradition in the literature of verifiable electronic voting. While almost every existing cryptographic voting protocol uses a BB as a central communication platform between the parties involved, almost no paper describing such a protocol gives a precise specification of the properties expected from the board. Usually, the existence of an appropriate BB is just taken for granted, but the BB itself remains a black box.

Given the importance of the bulletin board concept in electronic voting, only a remarkably small number of specific papers devoted to the problem of specifying and implementing a BB exists. Peters was one of the first to suggest such a specification and solution (Peters, 2005). His focus was on making the bulletin board robust against failures or attacks, using multiple peers and protocols from the multi-party computation literature. Heather and Lundin (2008; Lundin and Heather, 2010) made some proposals to ensure the append-only property and to solve the resulting conflicts with the robustness property. Some reports on corresponding implementations were published later (Krummenacher, 2010; Beuchat, 2012). Another description of a practical BB implementation is included in the report about the voting system used in the state of Victoria, Australia (Burton et al., 2012). In a follow-up paper, Culnane and Schneider (2014) proposed a robust algorithm for a peered bulletin board and verified its correctness formally.

2. Broadcast Channel with Memory

Many cryptographic voting protocols in the literature use the concept of a broadcast channel with memory to achieve universal verifiability. The BCM is used by the involved parties to exchange public data during the execution of the protocol. Unfortunately, most authors assume the existence of a BCM without providing a detailed definition of what a BCM is and without listing its properties. The lack of proper definitions is a problem for the general understanding of the cryptographic protocols and for analyzing their security properties. In this section, based on the notion of a distributed system, we give formal definitions of various types of channels, including broadcast channels and broadcast channels with memory. In Section 2.2, we sketch an exemplary cryptographic voting protocol, in which public data is exchanged over a broadcast channel with memory, and we discuss the importance of this channel to achieve universal verifiability.

2.1. Distributed Systems and Channels

According to (Buttyán, Staamann, and Wilhelm 1998), let a distributed system (Ω, Γ) consist of a finite set of parties $\Omega = \{p_1, \dots, p_n\}$ and a finite set of channels $\Gamma = \{c_1, \dots, c_m\}$. The parties of a distributed system exchange messages $m \in \mathcal{M}$ over the available channels, where $\mathcal{M}$ represents the message space of all possible messages, for example $\mathcal{M} = \{0,1\}^*$. The type, content, and order of the messages is specified by the cryptographic protocol, which aims at achieving some security (and other) objectives in the context of a service or application provided by the distributed system. For achieving these objectives, it is assumed that some channels offer properties such as authenticity or confidentiality. In this section, we keep channels offering such properties on an abstract level, so that we can introduce their properties without addressing the problem of how to implement them.

As a general simplifying and idealizing rule, we assume that channels are noiseless and that they possess unlimited capacity. This means that no message sent over a channel is lost or modified during transmission and that the transmission is instantaneous, regardless of the message size. We also assume that messages sent over a given channel are totally ordered, i.e., the channel does not allow sending two messages m_1 and m_2 at the exact same point in time. A channel satisfying all of these properties is called ideal. Subsequently, we consider distributed systems consisting of ideal channels only.

Definition 1. A (ideal) channel $c \in \Gamma$ of a distributed system (Ω, Γ) is defined by a sender domain $S_c \subseteq \Omega$ (the parties that can send messages over c), a receiver domain $R_c \subseteq \Omega$ (the parties that can receive messages over c), and a message space $\mathcal{M}_c \subseteq \mathcal{M}$ (the messages that can be transmitted over c). If $s \in S_c$ transmits $m \in \mathcal{M}_c$ over c to R_c, then every $r \in R_c$ receives m instantaneously at the moment when m is sent. Parties $p \in \Omega \backslash R_c$ not from the receiver domain can observe the transmission of m, but can not learn any information about m itself. On the other hand, parties $p \notin \Omega$ not belonging to the distributed system do not have access to the channels and therefore cannot observe the transmissions.

This general definition of an ideal channel includes several useful limiting cases, which are important in cryptographic protocols. We call $c \in \Gamma$ a public channel, if $S_c = \Omega$. This means that every party in the system can send messages over c. Similarly, c is called broadcast channel, if $R_c = \Omega$. In this case, every message transmitted is received by all parties in the system. If $S_c = \{s\}$ consists of a single sender $s \in \Omega$, then c is an authentic channel. Receiving m over such an authentic channel guarantees that s is the author of m. Similarly, if $R_c = \{r\}$ consists of a single receiver $r \in \Omega$, then c is a confidential channel. In this case, the channel guarantees that no party other that r learns m. If the sender and receiver domains are identical, i.e., if $S_c = R_c$, we speak of a closed group channel. In this case, every member of the closed group can send and receive messages over c.

Some of the above properties are mutually exclusive. For instance, a broadcast channel cannot be confidential and a public channel cannot be authentic (except for $|\Omega| = 1$). On the other hand, there are several useful combinations that are very common in cryptographic protocols. Most importantly, if c is authentic and confidential at the same time, i.e., if both $S_c = \{s\}$ and $R_c = \{r\}$

consist of a single party only, it is called a secure channel (note that $s = r$ is not excluded by this definition). A secure channel c is called untappable in the special case of $\Omega = \{s, r\}$. This implies that no other party can observe the transmission of messages between s and r. For other meaningful combinations of the above properties (public broadcast channel, authentic broadcast channel, public confidential channel), we do not introduce specific terms.

2.1.1. Channel with Memory

In our ideal model of noiseless channels with unlimited capacities, we assume that every submitted message reaches every receiver from the receiver domain instantaneously, independently of the receiver's actual availability and capacity to process the incoming message. In a non-ideal setting, receivers might not always be capable of processing the messages when they arrive. They might even miss some incoming messages entirely. In cryptographic protocols, in which broadcast channels are used to spread information to everyone, this imperfection can cause complicated coordination problems.

To allow a receiver to recover from messages lost during the protocol execution, we introduce the concept of a channel with memory. The idea is that the transmission of a message $m \in \mathcal{M}_c$ over a channel $c \in \Gamma$ is performed by two operations $s\colon \mathsf{Send}_c(m)$ and $r\colon \mathbf{M} \leftarrow \mathsf{Receive}_c()$. The former is invoked by the sender $s \in S_c$ and the latter by the receiver $r \in R_c$. A channel with memory guarantees that the messages and the order in which they have been sent are stored and never lost. For this, the channel maintains an internal state, called channel history $\mathbf{M}$, which is updated each time a new message is sent. This idea is further formalized in the following definition.

Definition 2. Let $c \in \Gamma$ be a channel of a distributed system (Ω, Γ) with sender domain $S_c \subseteq \Omega$ and receiver domain $R_c \subseteq \Omega$. We call c channel with memory if every $s \in S_c$ can perform the operation $s\colon \mathsf{Send}_c(m)$ to send an arbitrary message $m \in \mathcal{M}_c$ over the channel and every $r \in R_c$ can perform $r\colon \mathbf{M} \leftarrow \mathsf{Receive}_c()$ to receive an ordered list $\mathbf{M} \in \mathcal{M}_c^*$ of all messages sent so far. In other words, for all possible sequences of send operations,

$$\begin{aligned} &s_1\colon \mathsf{Send}_c(m_1),\\ &\quad\vdots\\ &s_t\colon \mathsf{Send}_c(m_t), \end{aligned}$$

performed by senders $s_1, \ldots, s_t \in S_c$ prior to performing $r\colon \mathbf{M} \leftarrow \mathsf{Receive}_c()$, we expect $\mathbf{M} = \langle m_1, \ldots, m_t \rangle$ to contain all messages in the given order.

The simplest way for a channel to achieve this property is to update an initially empty list $\mathbf{M} \leftarrow \langle\rangle$ by $\mathbf{M} \leftarrow \mathbf{M} \parallel m$ each time a new message $m \in \mathcal{M}_c$ arrives. We assume that a channel with memory has unlimited capacity, i.e., its history M can get arbitrarily large.

The concept of a channel with memory applies to all specific channel types described before. For example, a broadcast channel with memory (BCM) gives every involved party full access to all the messages sent over this channel. In a confidential channel with memory, this access is restricted to a single receiver. In an authentic channel with memory, it is guaranteed that every

message m_i included in the channel history **M**, has been sent by a single sender. Finally, in a public channel with memory, nothing is known about the senders of the messages m_i included in **M**.

2.1.2. Bundled Channel

In some situations, cryptographic protocols require the existence of many similar channels, for example, multiple authentic channels from different senders to the same receiver domain. Such a set of channels can be regarded as a single bundled channel, which inherits some properties from its components. The purpose of introducing bundled channels is to facilitate the description of cryptographic protocols with a large number of parties. In the following definition, we distinguish two opposite types of bundled channels.

Definition 3. For a given distributed system (Ω, Γ), a bundled input channel of size r is a non-empty set of channels $C = \{c_1, \dots, c_r\} \subseteq \Gamma$ with a common receiver domain $R_C = R_{c_i}$, for all $1 \leq i \leq r$. The bundled sender domain is defined as $S_C = \cup_{i=1}^{r} S_{c_i}$ and the bundled message space as $\mathcal{M}_C = \cap_{i=1}^{r} \mathcal{M}_{c_i}$. To invoke the transmission of a message $m \in \mathcal{M}_C$ over C, a sender $s \in S_C$ selects all channels $c_i \in C$ satisfying $s \in S_{c_i}$ for the actual transmission of m. Similarly, a bundled output channel of size r is a non-empty set of channels $C = \{c_1, \dots, c_r\} \subseteq \Gamma$ with a common sender domain $S_C = S_{c_i}$, for all $1 \leq i \leq r$. The bundled receiver domain is defined as $R_C = \cup_{i=1}^{r} R_{c_i}$ and the bundled message space as $\mathcal{M}_C = \cap_{i=1}^{r} \mathcal{M}_{c_i}$. To invoke the transmission of a message $m \in \mathcal{M}_C$ over C, a sender $s \in S_C$ selects all channels $c_i \in C$ for the actual transmission of m. In both cases, all receivers $r \in R_C$ receive the messages instantaneously over all available channels.

If the sender and receiver domains of all channels of a given set $C = \{c_1, \dots, c_r\} \subseteq \Gamma$ are identical, i.e., if $S_C = S_{c_i}$ and $R_C = R_{c_i}$ holds for all $1 \leq i \leq r$, then two concepts from the above definition merge into one. In this case, we call C simply a bundled channel. Note that every bundled (input or/and output) channel is a channel on its own. It is therefore possible to bundle channels recursively.

By applying the concept of a bundled input channel C to channels with memory, we propose to extend our definition of a channel with memory in order to keep track of the sender domains of every message sent over the bundled channel. Otherwise, useful information about the potential senders of the messages from the channel history **M** would be lost. We also propose that such a C keeps track of the global message order over all its channels, which is something that cannot be derived from the individual channel histories alone. In the following definition, $C_s = \{c_i \in C : s \in S_{c_i}\} \subseteq C$ denotes the set of channels available for a sender $s \in S_C$.

Definition 4. A bundled input channel with memory of size r is a bundled input channel $C = \{c_1, \dots, c_r\} \subseteq \Gamma$ consisting of r channels with memory c_i. To transmit a message $m \in \mathcal{M}_C$ over C to the receivers, a sender $s \in S_C$ of the bundled sender domain $S_C = \cup_{i=1}^{r} S_{c_i}$ performs the operation $s: \mathsf{Send}_C(m)$, which invokes $s: \mathsf{Send}_{c_i}(m)$ for all $c_i \in C_s$. If

$$\begin{aligned} &s_1: \mathsf{Send}_C(m_1), \\ &\quad\vdots \\ &s_n: \mathsf{Send}_C(m_t), \end{aligned}$$

is the sequence of all send operations performed by senders $s_1, \ldots, s_t \in S_C$ in the given order, then calling $(\mathbf{M}, \mathbf{S}) \leftarrow \mathsf{Receive}_C()$ returns two sequences $\mathbf{M} = \langle m_1, \ldots, m_t \rangle$ and $\mathbf{S} = \langle S_1, \ldots, S_t \rangle$, where $S_i = \cap_{c \in C_{s_i}} S_c$ denotes the set of possible senders of message m_i.

There are two important special cases of a bundled input channel with memory. The first one appears if C consists of broadcast channels, which implies that C itself is a broadcast channel with receiver domain $R_C = \Omega$. In this case, we simply speak of a bundled broadcast channel with memory (BBCM). This is the type of channel that we think is most useful in cryptographic voting protocols. It keeps track of every message, the potential senders of every message, and the global order over all messages sent over this channel. The current state of this information is accessible to every party in the system at any time.

The second special case appears if all channels $c_i \in C$ are authentic. This implies that each entry of the resulting sequence $\mathbf{S} = \langle \{s_1\}, \ldots, \{s_t\} \rangle$ is a singleton set $\{s_i\}$, i.e., the channel keeps track of the actual sender s_i of the messages m_i in this case. Such a bundled input channel with memory is called authentic. If some channels in C are authentic and some are not authentic, then C is called partially authentic. In a partially authentic bundled broadcast channel with memory C, some parties can broadcast their messages in an authentic and some on an unauthentic way over C. This is what is often required in a cryptographic voting protocol.

2.2. Using Channels in a Voting Protocol

To illustrate the application of different channels in a cryptographic protocol, we introduce a simplified cryptographic voting protocol, which contains many elements of real voting protocols from the literature. This protocol is executed by a distributed system (Ω, Γ) consisting of an election administrator E, some trusted election authorities $A_1, \ldots, A_s$, some eligible voters $V_1, \ldots, V_n$, and some external (non-eligible) verifiers $W_1, \ldots, W_m$. Therefore,

$$\Omega = \{E, A_1, \ldots, A_s, V_1, \ldots, V_n, W_1, \ldots, W_m\}$$

denotes the set of all parties involved in the protocol. The election administrator is responsible for registering the voters and for defining the context (title, period, voting rules, candidate list, voter roll, etc.) of the election, whereas the trusted authorities are responsible for some cryptographic tasks such as mixing and decrypting the list of encrypted votes and for sharing corresponding keys. Note that if a real-world person acts as an election authority and at the same time is an eligible voter, or if a real-world person is both a voter and a verifier, we consider them as two separate parties with different roles in the distributed system.

The next step is to define the set of channels Γ and their types. First, we assume that secure channels $sc_1, \ldots, sc_n$ exist between the voters and the election administrator for the registration process, i.e., $S_{sc_i} = \{V_i\}$ and $R_{sc_i} = \{E\}$ are the sender and receiver domains of these channels for $1 \leq i \leq n$. Second, we assume that an authentic broadcast channel ac_0 is available to the election administrator and that authentic broadcast channels $ac_1, \ldots, ac_s$ are available to the trusted authorities. They use them to publish the election definition and the results of their computations. $S_{ac_0} = \{E\}$ and $S_{ac_i} = \{A_i\}$, for $1 \leq i \leq s$, are the sender domains and $R_{ac_i} = \Omega$, for $0 \leq i \leq s$, are the

receiver domains of these channels. Finally, for casting their votes, we assume that voters have access to a public broadcast channel bc with $S_{bc} = R_{bc} = \Omega$. To sum up,

$$\Gamma = \{sc_1, \ldots, sc_n, ac_0, \ldots, ac_s, bc\}$$

is the set of channels we assume to exist in the protocol. The execution of the protocol is divided into the following consecutive phases:

Registration

1) Each voter V_i generates private/public election credentials and sends the public credential over the channel sc_i to the election administrator.

Election Preparation

2) The election administrator broadcasts the election definition (title, period, voting rules, candidate list) over the channel ac_0.
3) The election administrator defines the electoral roll and broadcasts the corresponding list of public election credentials over the channel ac_0.
4) Each trusted authority A_i generates shares of a private/public encryption key and broadcasts the public share over the channel ac_i.

Vote Casting

5) The voters encrypt their votes and broadcast corresponding ballots over the channel bc. The ballot includes some cryptographic proofs that the encrypted vote is well formed and that it is properly linked to one of the public credentials.

Tallying

6) The election administrator checks each ballot for validity, selects the encrypted vote from each valid ballot, and broadcasts the resulting list of encrypted votes over ac_0.
7) The trusted authorities perform the anonymization and decryption of the list of encrypted votes in a distributed manner and generate corresponding cryptographic proofs. These proofs, together with the list of decrypted votes, are broadcast over ac_i. The election result follows from the decrypted votes.

Verification

8) By checking all cryptographic proofs included in the ballots and obtained from the authorities during tallying, the verifiers re-calculate the election result.

The above protocol description relies on the assumption that all channels are ideal. It is therefore assumed that all involved parties receive every message transmitted over the broadcast channels. Since this ideal functionality is difficult to achieve in any practical setting, we replace the broadcast channels by broadcast channels with memory. Furthermore, because broadcast channels can always be bundled, we introduce a bundled broadcast channel with memory $C = \{ac_0, \ldots, ac_s, bc\}$ and replace Γ by $\Gamma' = \Gamma \cup \{C\}$. Consequently, we assume that C is used in Steps 2 through 6 of the above protocol to broadcast respective messages. In each case, broadcasting is invoked by $\mathsf{Send}_C(m)$. By calling $(\mathbf{M}, \mathbf{S}) \leftarrow \mathsf{Receive}_C()$ at any desired point in time, every party in the system has permanent access to the current state of the global channel history.

Note that C is partially authentic, which implies that messages from the administrator or the trusted authorities can be attributed unambiguously.

3. Bulletin Board in Electronic Voting

The concept of a bundled broadcast channel with memory, as introduced in the previous section, is an idealized theoretical construct, for which no one-to-one practical implementation exists in the real world. For this reason, cryptographic voting protocols that require such a channel need a substitution that provides an equivalent functionality and similar guarantees. Knowing that the exact same properties of an ideal broadcast channel with memory can at best be approximated by a practical implementation, designing such a substitution is a very delicate problem on its own. A common approach in the literature is to add one or multiple additional parties offering the service of a bulletin board to the other parties of the distributed system. We call them bulletin board parties and include them in an extended list of parties $\Omega' = \Omega \cup \{B_1, \ldots, B_r\}$. To substitute a broadcast channel with memory, it is necessary that all parties have access to a channel that connects them with the bulletin board parties for sending and receiving messages. The bulletin board parties themselves may be mutually connected over additional (authentic, confidential, or secure) channels to coordinate their current state of memory.

In this section, we first discuss the general objective and functionality of a BB. We also give a list of properties that are crucial for using a BB in a cryptographic voting protocol. To emphasize the consequences of using a BB with such properties instead of an ideal BBCM, we discuss some operational and organizational problems that might arise in a voting protocol and how these problems might be solved with the BB.

3.1. Basic Functionality and Properties

The general objective of a BB is to substitute a BBCM in a cryptographic protocol and to simulate its functionality and properties. Some properties such as the unlimited capacity of an ideal BBCM are obviously hard to imitate by real-world machines and network connections with limited capacity. Therefore, providing a service with unlimited capacity cannot be the objective of a BB, but it should at least offer protective measures to prevent the flooding of the service. Similarly, instantaneous message transmission is impossible in a real-world setting, in which distant network nodes and heavy network traffic may cause delayed message delivery. However, the BB should at least guarantee a unique message order over all incoming messages, which for instance can be realized with the aid of a trusted timestamping service.

Since a BB is a service that is intended to substitute a BBCM, it must provide a similar interface to its users. In a protocol with a BBCM, the parties can send and receive messages using the channel operations $\mathsf{Send}_C(m)$ and $\mathsf{Receive}_C()$, respectively. To avoid ambiguities, we denote corresponding BB operations by $\mathsf{Post}(m)$ and $\mathsf{Get}()$, and by calling the former, we say that a party posts m to the BB, which then publishes m. Clearly, the properties of these operations have to approximate the properties of the BBCM as precisely as possible. Very informally, we would expect that at least the following conditions always hold:

- All parties can call Post(m) and Get() at any desired time.
- Calling Get() always returns all messages posted so far.
- A unique message order can be derived from the result of calling Get().

A BB satisfying these basic conditions is commonly called append-only. In the presence of cheating bulletin board parties, these properties are difficult to achieve. Solving this problem under reasonable assumptions has been the focus in the existing BB literature (Peters 2005; Heather and Lundin 2008; Krummenacher 2010; Beuchat 2012; Burton et al. 2012; Culnane and Schneider 2014).

Because the BB is a service, not a channel, it can provide additional functionality to its users. For example, it can separate the data of different applications by logically grouping the incoming messages, and it can support more sophisticated Get operations, which returns only the messages of a specific logical group. Such additional features may help to overcome some of the intrinsic limitations of a BB implementation in comparison with the ideal BBCM model. In Section 4, we will specify a possible BB implementation and give a larger list of such additional features.

3.2. Operational Problems

Here we list problems that may arise when using a BB in an electronic voting protocol and that are normally not addressed by the protocol.

Conflicting Messages

The cryptographic voting protocol might need to prevent the involved parties from submitting a particular message multiple times, possibly with different and conflicting content. One of the simplest solutions to this problem is to declare the first message of a given type and from a given party to be the one that counts. In this case, the board may simply deny all subsequent messages from the same party. On the other hand, if the strategy is to count the last message, then the BB needs to memorize the order of the published messages based on their time of arrival.

Malformed Messages

In a cryptographic protocol, each posted message has a certain structure and content. In case a party sends an incomplete or malformed message—unwillingly or on purpose—, the BB needs a strategy for dealing with it. The simplest strategy is to reject malformed message and deny their publication, but for this the BB must be capable to detect them.

Replayed Messages

As messages already published on the BB can be read by anyone, the protocol needs to prevent copying of an existing message from someone else and sending it again in the other person's name. If vote updating is permitted in an election, this can be exploited for cancelling an updated vote by simply submitting the first vote a second time. To prevent this, the BB must be able to verify the freshness of the message. Again, there are multiple strategies for avoiding such problems, for example, by making each post dependent on its predecessors.

Early and Late Messages

Every cryptographic voting protocol specifies some sort of election period. Only ballots posted during this period are accepted in the final tally. For ballots arriving too early or too late, the protocol needs a strategy of handling them. The simplest general solution is to let the BB reject all messages not arriving within the bounds of a specified period of time. Since rejecting messages is a delicate issue, especially in case of ballots in an election, they could also be published together with a timestamp. Generating unforgeable and accurate timestamps is, however, a difficult problem on its own.

Board Flooding

Depending on the protocol's strategy of dealing with conflicting, malformed, replayed, and early/late messages, the involved parties may be allowed to post arbitrarily many messages. This could be exploited by someone, who wants to disrupt the election. By sending a huge number of messages with unrelated content, the board can be flooded with messages until its capacity is reached. This problem gets even worse, if the BB accepts messages from parties outside the protocol. Note that this is not a problem for an ideal BBCM with unlimited capacity.

Authentication

In a protocol, such as the one sketched in the previous section, the BBCM is composed of one or multiple authentic channels. Translating such a partially authentic BBCM into a BB implies that the board can authenticate the senders of certain messages. The necessary infrastructure for providing secure authentication is not defined by the protocol, but needs to be specified carefully in an actual implementation.

Undeniable Receipts

In a protocol that relies on broadcasting, the BBCM ensures a reliable transmission to every receiver at any future point in time. This is not necessarily the case with a BB, since cheating bulletin board parties could try to suppress messages from being delivered. The BB should therefore issue an undeniable receipt to confirm the publication of each posted messages. In a voting protocol, voters could then keep a copy of this receipt to verify that their vote has been tallied correctly. In an actual implementation, the details of such undeniable receipts need to be specified.

Consistent Views

To verify the outcome of an election, the exact set of all published messages needs to be known to the verifiers. Clearly, different verifiers will only come to a consistent conclusion, if they all obtain the same view of this set. The BBCM provides consistent views by definition, but a cheating BB could easily respond to different `Get()` operations with different sets of messages. Measures to avoid such misbehavior need to be implemented in a practical BB.

3.3. Organizational Problems

The following organizational problems may occour when running an electronic voting system.

Extending the Election Period

Due to unexpected circumstances like power outages, network or server problems, or misinformation, it might be necessary to readjust the election period during an election. The protocol needs to explicitly include a procedure for dealing with such an exceptional situation. In a proper solution, the election administrator must announce the extended period on the BB. This implies that the above-mentioned strategy of dealing with late messages must be adjusted accordingly.

Multiple Elections

In a productive environment, setting up the BB for every election is not very practical. The board should therefore be designed to support multiple elections, for example by providing designated areas for different elections, which are strictly separated.

Simultaneous Elections

With a BB supporting multiple elections, it might happen that some elections run at the same time, possibly with different administrators, trusted authorities, or voters. Handling multiple elections simultaneously is another desirable property for a practical BB that needs to be considered in its design.

Closing an Election

At the end of an election, when the final result has been published and accepted, adding further messages to the election data should be prohibited. The BB should therefore provide a mechanism for closing an election by locking up the final state of the election data and by making this visible to the public.

Archiving an Election

In the strict sense of the word, an append-only BB must never allow the deletion of some of its contents. On the other hand, keeping the election data from all the elections in the past may not really be necessary or meaningful. The BB should therefore allow the removal of the data of an entire election as soon as the public has verified and accepted the final election result. To archive the data for future use, it can possibly be moved to another place that does not have the debit the capacity and resources of the bulletin board.

4. Implementing a Bulletin Board

To address the aforementioned problems in a uniform way, we require a flexible formal definition of what a BB actually is and how it works. Clearly, its main purpose is to store the set of published messages and allow users to retrieve them. As additional board properties may require some kind

of metadata to be added to each message, let $p = (m, \alpha, \beta)$ be a container for a message $m \in \mathcal{M}$ and its metadata. Furthermore, we differentiate the metadata included in such a post p by its origin as it can be added either by the author of the message (the user) or by the board, and use different symbols $\alpha \in \mathcal{A}$ and $\beta \in \mathcal{B}$, respectively. We will model α and β as lists of attributes (called user attributes and board attributes, respectively). For a set of indices I, we write α_I and β_I for selecting corresponding sub-lists of attributes from α and β.

In this extended model, the purpose of the BB is to store a set of posts rather than a set of messages. We use $\mathcal{P}_t$ to denote this set at some given point in time t. This set represents the board's internal state (we will later omit the index t and simply write $\mathcal{P}$ for the board's internal state). Note the important subtlety of defining $\mathcal{P}_t$ as a set and not as a list, even though the BB has the objective to provide a total order for the messages. There are two reasons to define this order over properties rather than over the data structure. First this allows for an easy integration of various solutions. We present one such solution in Section 4.3. A second solution when retrieving a subset $R \subset \mathcal{P}$ allows solutions that provide for every $p \in R$ the absolute position in $\mathcal{P}$. The objective is that no message can be deleted or modified can be expressed with the above notation by $\mathcal{P}_t \subseteq \mathcal{P}_{t'}$ for all $t' \geq t$.

In the remainder of this section, we first introduce the generic interface consisting of two basic operations for posting and retrieving messages. We then define multiple BB properties, which may help solve the problems identified in the previous section. The solution proposed for every property is realized within the boundaries of the generic interface by corresponding user and board attributes. With the distinction between properties for structuring the board content, keeping track of the board's history, and ensuring the authenticity and integrity of the posts.

4.1. Basic Operations

In the previous section, we informally introduced the two basic operations `Get` and `Post` that any BB should provide in its public interface. Our goal now is to specify them more precisely. The goal is to design the interface in a generic way so that we can change the properties of the board without changing the signature of the operations. In principle, a BB could support more operations such as `Update` or `Delete`, but these are in contradiction with the objectives of the service. Non-public operations for setting up and managing the board are not part of the public interface.

When publishing a message $m \in \mathcal{M}$, we said that the user will also provide some user attributes $\alpha \in \mathcal{A}$. Upon receiving m and α, the BB might do some checks to validate the post. In case a check fails, an error message $\perp$ is returned and the procedure aborts. Otherwise, some board attributes $\beta \in \mathcal{B}$ are generated and the post $p = (m, \alpha, \beta) \in \mathcal{M} \times \mathcal{A} \times \mathcal{B}$ is formed. We model both $\alpha = (\alpha_1, \ldots, \alpha_u)$ and $\beta = (\beta_1, \ldots, \beta_v)$ as corresponding lists of values $\alpha_i \in \mathcal{A}_i$ and $\beta_i \in \mathcal{B}_i$ without further specifying the sets $\mathcal{A}_i$ and $\mathcal{B}_i$. Note that not necessarily all combinations of attributes might be permitted, which is why we define $\mathcal{A} \subseteq \mathcal{A}_1 \times \cdots \times \mathcal{A}_u$ and $\mathcal{B} \subseteq \mathcal{B}_1 \times \cdots \times \mathcal{B}_v$ as subsets of the corresponding Cartesian products.

To conclude the procedure of posting something to the BB, the board's current state is updated to $\mathcal{P} \leftarrow \mathcal{P} \cup \{p\}$ and β is returned to the user. Note that by receiving β as a response of posting

(m, α), the user gets in possession of the full post p. The following signature summarizes the Post operation:

$$\beta \vee \perp \leftarrow \mathsf{Post}(m, \alpha).$$

To obtain the simplest possible operation for retrieving the board's current content, we could define it as $\mathcal{P} \leftarrow \mathsf{Get}()$. The users would then always obtain every single post contained in $\mathcal{P}$, even if only certain posts are of interest. In a productive environment, where $\mathcal{P}$ could grow into a very large set, this solution might not be very practical. Therefore, we let the user define a query $Q \subseteq \mathcal{M} \times \mathcal{A} \times \mathcal{B}$, which is applied as a filter to the elements of $\mathcal{P}$. Therefore, the result of the query is the set $R = \mathcal{P} \cap Q$. Note that an unconstrained query $Q = \mathcal{M} \times \mathcal{A} \times \mathcal{B}$ results in returning the full set $\mathcal{P}$ as above.

In addition to returning R, the board might also produce and return some metadata $\gamma \in \mathcal{C}$ about the result of the query. As above, the metadata is modelled as a list $\gamma = (\gamma_1, \ldots, \gamma_w)$ of attributes $\gamma_i \in \mathcal{C}_i$ without further specifying the sets $\mathcal{C}_i$. We call them result attributes. The following signature summarizes the Get operation:

$$R, \gamma \leftarrow \mathsf{Get}(Q).$$

For better readability, we sometimes write $\mathcal{P}_Q$ for the resulting subset of posts $R = \mathcal{P} \cap Q$. For a query restricting only the i-th user attribute to a single value $\alpha_i \in \mathcal{A}_i$ or to a subset of values $A_i \subseteq \mathcal{A}_i$, we use the simplified notation $Q = \langle \alpha_i \rangle$ and $Q = \langle A_i \rangle$, respectively. Similarly, we write $\langle \alpha_i, \alpha_j \rangle = \langle \alpha_i \rangle \cap \langle \alpha_j \rangle$ or $\langle A_i, A_j \rangle = \langle A_i \rangle \cap \langle A_j \rangle$ for restrictions on multiple user attributes i and j. The same notational convention can be applied to board attributes or to mixed restrictions on user and board attributes.

4.2. Structuring the Board Content

The following three properties help structuring the content of the BB and in doing so allow to reduce the communication costs.

Property 1: Sectioned

A public bulletin board is called sectioned, if it consists of multiple equally shaped sections. The goal of a sectioned bulletin board is to separate unrelated messages into logically independent units. Let $\mathcal{S}$ denote the set of available sections. To enable the dispatching of an incoming post into the right section, the author must provide the section $s \in \mathcal{S}$ as a user attribute. A post containing an invalid section $s \notin \mathcal{S}$ is rejected by the board. In an election application, it would be natural to define individual sections for the data of each election. Therefore, this property addresses the problem of using a BB for multiple, possibly simultaneous, elections (see Section 3.3).

Property 2: Grouped

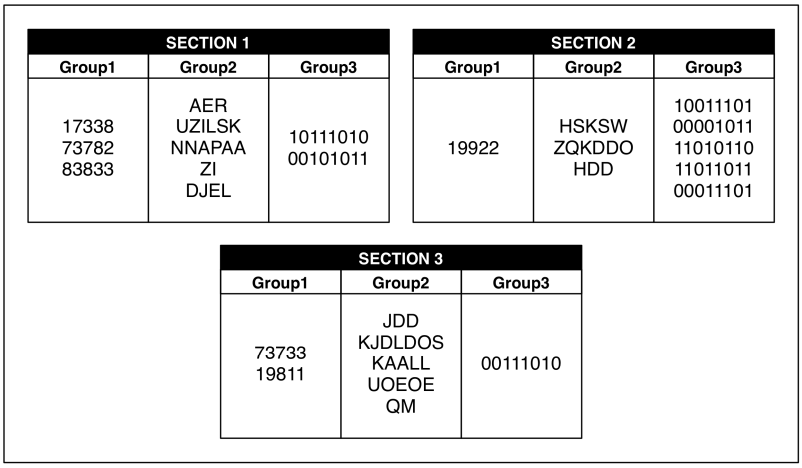
In a grouped bulletin board, messages are organized into groups. Typically, messages contained in the same group are similar in shape and content. Let $\mathcal{G}$ be the set of available groups. When posting a message, the author must indicate the group $g \in \mathcal{G}$ to which the message belongs

as a user attribute. A post containing an invalid group $g \notin \mathcal{G}$ is rejected by the board. Note that groups are independent of sections, i.e., every section in a sectioned board consists of the same set of groups $\mathcal{G}$. show an example of a board with three sections and three groups. In an election application, we would define individual groups for every different post in the protocol, for example for the list of candidates or the ballots. Introducing groups in a BB used for elections does not directly address one of the problems listed in Section 3, but it is a prerequisite for the following property.

Property 3: Typed

A grouped bulletin board is called typed, if each group $g \in \mathcal{G}$ defines its own set $\mathcal{M}_g \subseteq \mathcal{M}$ of valid messages. $\mathcal{M}_g$ is called type of g. In a typed board, an incoming message m for group g is accepted only if $m \in \mathcal{M}_g$, while all other messages $m \notin \mathcal{M}_g$ are rejected. The example of shows a typed board with different types of messages for each group, for example $\mathcal{M}_{\text{Group1}} = \{0, \dots, 9\}^5$, $\mathcal{M}_{\text{Group2}} = \{A, \dots, Z\}^*$, and $\mathcal{M}_{\text{Group3}} = \{0,1\}^8$. By using a typed BB for an election, we address the problem of malformed messages (see Section 3.3).

Figure 1: Example of a structured bulletin board with three sections, three groups, corresponding types, and some messages.



4.3. History of Board Content

Here we introduce three properties that help to establish the history of the BB.

Property 5: Ranked

In a ranked bulletin board, the posts $\mathcal{P}_{\langle s \rangle}$ published for a given section $s \in \mathcal{S}$ are ranked according to their time of arrival.[1] This can be achieved by adding a sequence number $i \in \mathbb{N}$ to each

[1] An unsectioned BB can always be considered as a board with a single section.

post, for example $i = |\mathcal{P}_{\langle s \rangle}|$ to obtain consecutive numbers 0,1,2, ... for each section. In our generic setting, the BB includes this number as a board attribute in β. Note that keeping track of the general message order over all board sections is not mandatory because we consider sections as logically independent units. In an election system with a BCM, knowing the message order is required. The total order can also be used for solving the problem of conflicting messages, for example, in case the voter submits multiple ballots (see Section 3.2). Recall that this is only one of many solutions to provide a total order of messages on the BB.

Property 6: Chronological

A bulletin board is called chronological, if a timestamp $t \in \mathcal{T}$ indicating the exact time an arrival is added to every incoming post. The BB is responsible for generating accurate timestamps and adding them as board attributes to β. Note that we cannot exclude that multiple posts receive identical timestamps, especially in case of a coarse time unit. A chronological BB is therefore not automatically ordered. In an election system, attaching timestamps to ballots is important to decide whether they have been received within the election period, or more generally to solve the problem of early or late messages (see Section 3.2).

Property 7: Interlinked.

A bulletin board is called interlinked, if every post in every section depends on all its predecessors in that section. More formally, let $p_i \in \mathcal{P}_{\langle s \rangle}$ denote the post with sequence number i in a given section $s \in \mathcal{S}$ and $\mathcal{P}_{\langle s,\{0,\ldots,i-1\}\rangle} = \{p_0, \ldots, p_{i-1}\}$ the set of predecessors of p_i in $\mathcal{P}_{\langle s \rangle}$. Interlinking p_i with all its predecessors can be achieved by applying some function H (typically a hash function) to $\mathcal{P}_{\langle s,\{0,\ldots,i-1\}\rangle}$. Another solution is to apply H only to the previous post p_{i-1}, which then creates indirect links to all predecessors of p_{i-1}. Such a construction is sometimes called a hash chain. In our generic setting, the resulting value $H_i = H(p_{i-1})$ can be included in $p_i = (m, \alpha, \beta)$, either as a user attribute in α or as a board attribute in β (an initial value H_0 is added to p_0). The first option corresponds to the construction proposed in Heather and Lundin (2008) for achieving the append-only property. It implies that no pair (m, α) will appear more than once on the board and thus eliminates the problem of replayed messages (see Section 3.2). However, due to coordination problems between users posting messages simultaneously, it is rather complicated to implement properly. In combination with other properties from the next subsection, an interlinked BB also helps addressing the problems of undeniable receipts and consistent views.

4.4. Authentication and Integrity

The two properties presented here help to ensure the authenticity and integrity of the messages on the BB.

Property 8: Access-Controlled

A bulletin board is called access-controlled if it provides an access-control mechanism that authenticates the author of a message and rejects the message if the user is not authorized. To enable the BB doing this check, we assume that a set $\mathcal{K}$ of public signature keys—one for each authorized user—is known to the board at every moment. This set is either static or dynamic. In

the static case, $\mathcal{K}$ is publicly known and cannot be changed, whereas in the dynamic case, $\mathcal{K} = K(\mathcal{P}_t, \alpha, \beta)$ is defined implicitly by a publicly known function K, which depends on the current board state $\mathcal{P}_t$ and the attributes included in the incoming post $p = (m, \alpha, \beta)$. The three arguments of K are optional, i.e., not all of them are relevant in every case. For p to be accepted by the board, the user's public key pk and a signature $S = \mathsf{Sign}_{sk}(m, \alpha_I)$ must be included as user attributes in α (we use α_I to denote the list of user attributes different from pk and S). The board can then perform the checks $pk \in \mathcal{K}$ and $\mathsf{Verify}_{pk}(m, \alpha_I; S)$ to decide whether p stems from an authorized user or not.

In electronic voting, our proposal of a dynamic set $\mathcal{K}$ may serve multiple purposes. For example, we can define a function K that allows the election administration or the trusted authorities to post exactly one message of a given type. Similarly, in a system that prohibits vote updating, we can give voters the right to submit exactly one ballot. For this, K must depend on $\mathcal{P}_t$ (to check if a message of the same type has been posted earlier by the same author) and on α (which contains the author's public key). This mechanism is therefore a solution for the problem of conflicting messages. Together with other measures against malformed or replayed messages, it also helps avoiding board flooding attacks (see Section 3.2).

Another possible application of a dynamic set $\mathcal{K}$ is to restrict the voter's right to submit ballots to the election period. In a chronological BB, every post contains a timestamp in the list of board attributes, which implies that in this case K must depend on β. Using such quantitative and temporal restrictions on the board's access rights, we can properly implement the process of closing an election (see Section 3.3). As soon as all access rights have been expended or have expired, the board's content reaches a final state. In a sectioned BB, a final state can be reached for each individual section. This is a precondition for archiving the data of an election after some time.

Property 9: Certified Publishing

A bulletin board offers certified publishing (Heather and Lundin 2008) if the board attests any response returned to a user with a digital signature. Depending on the operation, we distinguish between two sub-properties: certified posting and certified reading. Upon receiving (m, α) from a user calling the Post operation, the board generates a signature $S_{\mathsf{Post}} = \mathsf{Sign}(m, \alpha, \beta_I)$ and adds S_{Post} as a board attribute to β. With β_I we denote the list of board attributes different from S_{Post}. Similarly, upon responding a user's query Q with the result R, the board generates a signature $S_{\mathsf{Get}} = \mathsf{Sign}(Q, R, \gamma_I)$ and adds S_{Get} together with a timestamp t as result attributes to γ. Again, γ_I denotes the list of result attributes different from S_{Get} (but including t). Recall that returning β and γ to the user is already part of the Post and Get operations in the generic interface.

In both cases, the user can check the validity of the signature using the board's public key. Note that in an interlinked BB, the signature S_{Post} is not only a receipt for the publication of the message, but also a commitment to the current content of the board. Similarly, each signature S_{Get} is a commitment of the board to its content at time t. By issuing such commitments with each accepted post and for each query, the board guarantees the consistency of its history and therefore the integrity of the stored data. In an election system, this is a precondition for offering consistent views of the election data to every verifier (see Section 3.2).

4.5. Putting Everything Together

To conclude this section, let us consider a bulletin board satisfying all the properties described above (the variant in which the hash chain is generated by the board). To post a message $m \in \mathcal{M}_g$ to the board, the user must provide a list of user attributes $\alpha = [s, g, pk, S]$ containing a section $s \in \mathcal{S}$, a group $g \in \mathcal{G}$, the user's public key $pk \in \mathcal{K}$, and a signature $S = \mathsf{Sign}_{sk}(m, [s, g])$ generated using the user's secret key sk. If the post is accepted, the board responds with a list of board attributes $\beta = [i, t, H_i, S_{\mathsf{Post}}]$ containing a sequence number $i \in \mathbb{N}$, a timestamp $t \in \mathcal{T}$, a hash value $H_i = H(p_{i-1})$, and a signature $S_{\mathsf{Post}} = \mathsf{Sign}(m, \alpha, [i, t, H_i])$:

$$[i, t, H_i, S_{\mathsf{Post}}] \vee \perp \leftarrow \mathsf{Post}(m, [s, g, pk, S])).$$

If a query Q is sent to the bulletin board, it responses with the result R and a list of result attributes $\gamma = [t, S_{\mathsf{Get}}]$ containing a timestamp $t \in \mathcal{T}$ and a signature $S_{\mathsf{Get}} = \mathsf{Sign}(Q, R, [t])$:

$$R, [t, S_{\mathsf{Get}}] \leftarrow \mathsf{Get}(Q).$$

5. Conclusion and Outlook

In this paper, we introduced a formal model for a broadcast channel with memory and defined its ideal functionality. We then showed how to apply this model to an exemplary cryptographic voting protocol and how to implement it in a non-ideal real-world setting. The additional parties needed in such an implementation provide a service called bulletin board. Based on an informal definition of this service and the exemplary cryptographic voting protocol, we encountered a number of operational and organizational problems that may arise and that need to be addressed in a BB implementation. Finally, we proposed a design for a BB with a generic interface, which helps to make implementations more flexible and adaptable to different needs.

In the future, we plan to expand the channel model with additional definitions and properties. Concerning the BB, we will try to combine our approach with existing techniques to guarantee robustness under minimal assumptions.

References

Beuchat, J. 2012. "Append-Only Web Bulletin Board." Master's thesis, Biel, Switzerland: Bern University of Applied Sciences.

Burton, C., C. Culnane, J. Heather, T. Peacock, P. Y. A. Ryan, S. Schneider, S. Srinivasan, V. Teague, R. Wen, and Z. Xia. 2012. "A Supervised Verifiable Voting Protocol for the Victorian Electoral Commission." In EVOTE'12, 5th International Workshop on Electronic Voting, edited by M. Kripp, M. Volkamer, and R. Grimm, 81–94. Lecture Notes in Informatics P-205. Bregenz, Austria.

Buttyán, L., S. Staamann, and U. G. Wilhelm. 1998. "A Simple Logic for Authentication Protocol Design." In CSFW'11, 11th IEEE Computer Security Foundations Workshop, 153–62. Rockport, USA.

Cohen, J. D., and M. J. Fischer. 1985. "A Robust and Verifiable Cryptographically Secure Election Scheme (Extended Abstract)." In FOCS'85, 26th IEEE Symposium on Foundations of Computer Science, 372–82. Portland, Oregon.

Cramer, R., R. Gennaro, and B. Schoenmakers. 1997a. "A Secure and Optimally Efficient Multi-Authority Election Scheme." In EUROCRYPT'97, 16th International Conference on the Theory and Application of Cryptographic Techniques, edited by W. Fumy, 103–18. LNCS 1233. Konstanz, Germany.

———. 1997b. "A Secure and Optimally Efficient Multi-Authority Election Scheme." European Transactions on Telecommunications 8 (5): 481–90.

Culnane, C., and S. Schneider. 2014. "A Peered Bulletin Board for Robust Use in Verifiable Voting Systems." Computing Research Repository arXiv:1401.4151.

Haenni, R., and R. E. Koenig. 2013. "A Generic Approach to Prevent Board Flooding Attacks in Coercion-Resistant Electronic Voting Schemes." Computers & Security 33: 59–69.

Hauser, S. and R. Haenni "A generic interface for the public bulletin board used in UniVote" In CeDEM'16, 6th International Conference for E-Democracy and Open Government, edited by P. Parycek, N. Edelmann 49-56. Krems, Austria

Heather, J., and D. Lundin. 2008. "The Append-Only Web Bulletin Board." In FAST'08, 5th International Workshop on Formal Aspects in Security and Trust, edited by P. Degano, J. Guttman, and F. Martinelli, 242–56. LNCS 5491. Malaga, Spain.

Krummenacher, R. 2010. "Implementation of a Web Bulletin Board for E-Voting Applications." Project Report, Switzerland: Hochschule für Technik Rapperswil (HSR).

Lundin, D., and J. Heather. 2008. "The Robust Append-Only Web Bulletin Board." Guildford, U.K.: University of Surrey.

Peters, R. A. 2005. "A Secure Bulletin Board." Master's thesis, Department of Mathematics; Computing Science, Technische Universiteit Eindhoven, The Netherlands.

About the Authors

Severin Hauser

PhD Student at the University of Fribourg and works in the Research Institute for Security in the Information Society (RISIS) at the Bern University of Applied Sciences as a researcher. His current research focuses on secure electronic voting in general and specifically the bulletin board.

Dr. Rolf Haenni

Professor in Computer Science at the Bern University of Applied Sciences (BFH) in Switzerland. He is a member of the E-Voting Research Group, which is part of the Research Institute for Security in the Information Society (RISIS). He received his PhD degree in Computer Science in 1996 from the University of Fribourg in Switzerland. He has been affiliated to the University of Los Angeles, the University of Konstanz in Germany, and the University of Bern in Switzerland. His research interests include cryptographic protocols, secure electronic voting, privacy enhancing technologies, trust management, uncertain reasoning, probabilistic logics, and graphical models.

